INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Vers des primitives propres en arithmétique des ordinateurs

Jean-Michel Muller

No 3755

Septembre 1999

THÈME 2

ISSN 0249-6399

Vers des primitives propres en arithmétique des ordinateurs

Jean-Michel Muller*

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arenaire

Rapport de recherche n° 3755 — Septembre 1999 — 8 pages

Résumé : La norme IEEE-754 consacrée à l'arithmétique virgule flottante spécifie le comportement des quatre opérations arithmétiques. Une spécification des fonctions élémentaires devrait voir le jour dans les années à venir. On s'intéresse dans cet article aux avantages que l'on peut tirer d'un système dont les "primitives numériques" sont complètement spécifiées.

Mots-clé : Arithmétique des ordinateurs, virgule flottante, fiabilité des logiciels.

(Abstract: pto)

* CNRS, Laboratoire LIP, projet CNRS-ENSL-INRIA ARENAIRE, Ecole Normale Supérieure de Lyon

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International: +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

Toward "Clean" Primitives in Computer Arithmetic

Abstract: The IEEE-754 floating-point standard gives a specification of the four basic arithmetic operations. The elementary functions should be standardized as well within a few years. In this paper, we present the various advantages of a completely specified arithmetic system.

Key-words: Computer arithmetic, floating-point arithmetic, software reliability.

1 Introduction

L'arithmétique des ordinateurs a connu un changement majeur en 1985, avec l'apparition de la norme IEEE-754 [2], qui spécifie les formats de représentation des nombres et les opérations arithmétiques en virgule flottante. Cette norme s'est rapidement imposée. Ses aspects les plus connus sont les formats de représentation (base 2, nombres de bits de mantisse et d'exposant, etc.), mais ses aspects les plus utiles concernent la gestion des arrondis et des exceptions. Concentrons nous sur les arrondis. La norme demande que l'utilisateur puisse choisir un mode d'arrondi "actif" parmi les 4 suivants : au plus près (mode par défaut), vers le haut (c'est-à-dire vers le plus petit nombre machine supérieur ou égal au résultat), vers le bas ou vers 0. Le résultat de l'une des 4 opérations arithmétiques ou de la racine carrée doit être celui qu'on obtiendrait si on faisait d'abord le calcul avec une "précision infinie" avant d'arrondir avec le mode d'arrondi actif. Cette exigence, appelée "arrondi correct", a de nombreuses conséquences :

- la portabilité des programmes numériques s'en trouve améliorée ;
- on peut construire des *algorithmes* et des *preuves* qui utilisent ces spécifications. Cela a par exemple été fait par Kahan et certains de ses élèves tels que Priest [11] ou Shewchuk [12] à l'*University of California at Berkeley*. Par la suite, je vais sur des exemples simples développer ce point ;
- en jouant sur les différents modes d'arrondi, il est possible de faire simplement de l'*arithmétique d'intervalles* ou de l'*arithmétique stochastique*. Ces deux arithmétiques servent à faire du contrôle d'erreur.

2 Quelques résultats utilisant des propriétés de la norme IEEE 754

2.1 Un exemple très simple

Considérons le petit programme suivant [6].

```
variables A, B: virgule flottante;
A := 1.0; B := 1.0;
tant que (A+1)-A-1 = 0 faire A := 2*A;
tant que (A+B)-A-B <> 0 faire B := B+1
```

Sur une machine dont l'arithmétique est en base β, et qui effectue les opérations avec arrondi correct, la valeur finale de `B` sera β. Pour prouver ceci, il est nécessaire d'utiliser le fait que l'arrondi est correct : si l'on suppose les opérations exactes, on

concluera que le programme se termine par un dépassement de capacité. Si l'on se contente de supposer que les opérations flottantes sont une approximation (à une certaine erreur relative près) des opérations exactes, on ne peut rien conclure.

2.2 Normalisation d'un vecteur, rotations

Kahan a montré [7] que sur toute machine compatible avec la norme IEEE, le calcul $Z = X/\sqrt{X^2+Y^2}$, en arrondi au plus près, où X et Y sont des nombres machine, donnera un résultat compris entre -1 et $+1$. Une telle propriété est importante. En arithmétique réelle exacte, elle est toujours vraie : un programmeur inexpérimenté pourra donc la considérer comme "intuitivement vraie" et calculer une fonction de Z qui n'est définie que si $|Z| \leq 1$. Sur une machine qui n'est pas compatible avec la norme IEEE, le comportement du programme risque d'être imprévisible. Un programmeur conscient du problème peut choisir d'effectuer des tests. Ceux-ci ne provoqueront pas d'erreurs mais seront inutiles sur une machine compatible avec la norme.

2.3 Addition "sans erreur"

On a parfois momentanément besoin de plus de précision que la précision usuelle des formats virgule flottante. En utilisant les propriétés de la norme IEEE-754, on peut montrer que des algorithmes très simples tels que celui donné ci-dessous fournissent des résultats corrects.

Théorème 1 (Dekker, 1971) *Si x et y sont des "nombres machine", avec $|x| \geq |y|$, alors sur un système conforme à la norme IEEE-754, et si aucun dépassement de capacité n'intervient, la suite de calculs [5]*

```
z := x+y;     yprime := z-x;     e := y-yprime
```

fournira le résultat exact de l'addition $x+y$ sous la forme de 2 variables z et e. On entend par là que $x+y = z+e$ exactement, que z est le résultat "normal" de l'addition flottante (suivant le mode d'arrondi actif), et que e est l'erreur de cette addition.

Des résultats similaires existent pour la multiplication. Shewchuk [12] et Daumas [4] ont généralisé ce type d'algorithme à la manipulation "d'expansions." Des algorithmes similaires peuvent être construits pour des arithmétiques sans arrondi correct, mais vérifiant des conditions plus "lâches." Ils sont plus complexes.

3 Le problème des fonctions élémentaires

Les implantations actuelles des fonctions élémentaires laissent parfois à désirer (voir par exemple [9], Table 8.1, page 147). Ceci est dû à l'absence de spécification de ces fonctions, elle même due au fait qu'il est difficile de garantir leur arrondi correct. A part dans des cas très simples on ne peut obtenir le résultat exact en effectuant un nombre fini d'opérations. On peut seulement en calculer une approximation. Le problème est de savoir si en arrondissant l'approximation, on obtiendra la valeur que l'on aurait en arrondissant le résultat exact. Supposons des mantisses de 24 bits (c'est le cas de la simple précision de la norme IEEE). Considérons le nombre x dont l'écriture binaire est $1.11001100111000110011001$. Son exponentielle est :

$$110.000011010011110100110\ 0\ 11111111111111111111111101100100001\ldots$$

ce qui est très proche du milieu de deux nombres machine consécutifs. Si l'on désire calculer e^x en arrondi au plus près, il faudra d'abord obtenir une approximation très précise de cette exponentielle, afin de savoir si elle est au-dessous ou au-dessus du milieu des nombres machine $A = 110.000011010011110100110$ et $B = 110.000011010011110100111$. Garantir l'arrondi correct des fonctions élémentaires revient à déterminer à quelle précision il faut les approcher pour que l'arrondi de l'approximation soit toujours égal à l'arrondi du résultat exact. Le projet CNRS-ENSL-INRIA Arenaire travaille sur ce thème [8], et les résultats obtenus sont consultables à la page web

`http://www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm`

Des exemples de « pires cas » trouvés, qui permettent de savoir avec quelle précision les calculs intermédiaires doivent être effectués sont:

$$2^{51} \times \exp\left(\frac{4714832272232 79}{2^{49}}\right) = 5203087862132336.4999999999999999818\ldots$$

et

$$2^{54} \times \sin\left(\frac{8980155785351021}{2^{54}}\right) = 861282010793742.9999999999999999999870\ldots$$

3.1 Division

Un autre aspect utile de l'arrondi correct est le suivant. On peut construire un algorithme de division qui fournira toujours un arrondi correct, en utilisant comme

primitive un opérateur qui fournit avec arrondi correct un résultat de la forme $ax + b$. Pour ceci, Cornea-Hasan, Golliver et Markstein [3] ont montré plusieurs résultats semblables au suivant. Notons $(ax + b)_{rn}$ l'arrondi au plus près de $ax + b$. Si le système virgule flottante utilisé à n bits de mantisse, on dira que x et y sont à une distance d'au plus un *ulp* (*Unit in the Last Place*) si $|x - y| \leq 2^{e-n+1}$, où e est tel que le plus grand (en valeur absolue) des deux nombres — appellons-le t — vérifie $2^e \leq t < 2^{e+1}$.

Théorème 2 *[3] Soit B un nombre machine. Si y est un nombre machine situé à une distance d'au plus un ulp de $1/B$, alors la suite de calculs*

$$z = (1 - By)_{rn} \qquad y' = (Y + zy)_{rn}$$

fournira un résultat y' qui est l'arrondi correct au plus près de $1/B$.

Les résultats montrés dans [3] ont permis à leurs auteurs de concevoir un algorithme de division. Des propriétés similaires ont été utilisées pour concevoir les algorithmes de division et de racine carrée du microprocesseur K7 d'AMD [10], ainsi que ceux du microprocesseur Power3 d'IBM [1].

Conclusion

J'espère avoir montré les avantages qu'apporte la norme virgule flottante IEEE-754. J'ai uniquement parlé de l'arrondi correct, mais la gestion des exceptions constitue également un gros progrès. Il reste à utiliser les propriétés de cette arithmétique dans les étapes de validation des programmes. On peut regretter que les constructeurs aient des interprétations légèrement différentes de la norme (les arithmétiques du monde PC et du monde SPARC diffèrent sur certains points), et que les concepteurs de compilateurs ne se soucient guère de rendre aisées des opérations telles que le changement de mode d'arrondi. De plus, les arithmétiques *Quick and Dirty* des années $60-70$ risquent de revenir à la mode, avec l'arrivée de processeurs dédiés au calcul rapide 3D sur lesquels un mot contiendra plusieurs flottants. Tous les constructeurs travaillent actuellement sur de tels processeurs.

Références

[1] R.C. Agarwal, F.G. Gustavson, et M.S. Schmookler. Series approximation methods for divide and square-root in the power3 processor. In I. Koren et P. Kornerup, editeurs, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*

(Arith-14, Adelaide, Australia, Avril 1999), pages 116–123. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[2] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[3] M. A. Cornea-Hasegan, R. A. Golliver, et P. Markstein. Correctness proofs outline for newton-raphson based floating-point divide and square root algorithms. In I. Koren et P. Kornerup, editeurs, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Arith-14, Adelaide, Australia, Avril 1999)*, pages 96–105. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[4] M. Daumas. Multiplications of floating-point expansions. In I. Koren et P. Kornerup, editeurs, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Arith-14, Adelaide, Australia, Avril 1999)*, pages 250–257. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[5] T.J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971.

[6] W. M. Gentleman et S. B. Marovitch. More on algorithms that reveal properties of floating-point arithmetic units. *Communications of the ACM*, 17(5):276–277, Mai 1974.

[7] W. Kahan. What can you learn about floating-point arithmetic in one hour? file cs267fp.ps. Version PostScript accessible électroniquement à http://http.cs.berkeley.edu/∼wkahan/ieee754status, 1996.

[8] V. Lefèvre, J.-M. Muller, et A. Tisserand. Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11), Novembre 1998.

[9] J.M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston, 1997.

[10] S.F. Oberman. Floating-point division and square root algorithms and implementation in the amd-k7 microprocessor. In I. Koren et P. Kornerup, editeurs, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Arith-14, Adelaide, Australia, Avril 1999)*, pages 106–115. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[11] D. M. Priest. Algorithms for arbitrary precision floating point arithmetic. In P. Kornerup et D. W. Matula, editeurs, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, pages 132–144, Grenoble, France, Juin 1991. IEEE Computer Society Press, Los Alamitos, CA.

[12] J.R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the 12th annual ACM symposium on computational geometry*, pages 141–150, 1996.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399